AD-A250 242

NTATION PAGE

Form Approved
OMB No. 0704-0188

1. A

T DATE: March 1992

3. REPORT TYPE AND DATES COVERED: Professional paper

4. TITLE AND SUBTITLE

THE DESIGN OF A REAL-TIME ADAPTIVE FILTER DEVELOPMENT SYSTEM

5. FUNDING NUMBERS

In house

6. AUTHOR(S)

T. R. Albert, M. D. Juniper, R. C. North and W. H. Ku

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC), Research, Development, Test and Evaluation Division (NRaD) San Diego, CA 92152-5000

Center for Ultra-High Speed Integrated Circuits and Systems
University of California, San Diego
La Jolla, CA 92093-0407

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC),
Research, Development, Test and Evaluation Division (NRaD)
San Diego, CA 92152-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper describes a hardware test platform designed to implement adaptive lattice filters in real-time. To achieve real-time processing speeds, algorithm complexity was accommodated by custom designing the computation engines with respect to the lattice data flow. Execution speeds of the computation engines were dramatically increased by a memory architecture that supports efficient addressing and by providing a floating-point ALU with numerous data paths and efficient implementation of division. Performance is further enhanced by pipelining multiple computation engines. In addition, the architecture is flexible enough to support other filter structures and to allow observation of filter variables as they adapt. With this system, various Adaptive Filter algorithms are being tested in real-time implementations of Adaptive Line Enhancers (ALEs) and Adaptive Noise Cancelers (ANCs) in order to characterize their performance and behavior, especially long term stability and the ability to track non-stationary signals.

Published in *Proceedings of the Asilomar Conference on Circuits, Systems, and Computers,* Nov. 1990.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
T. R. Albert	(619) 553–1675	Code 632

The Design of a Real-Time Adaptive Filter Development System

Terence R. Albert and Michael D. Juniper
U.S. Naval Ocean Systems Center
San Diego, CA 92152-5000

Richard C. North and Walter H. Ku
Center for Ultra-High Speed Integrated
Circuits and Systems
University of California, San Diego
La Jolla, CA 92093-0407

Abstract

This paper describes a hardware test platform designed to implement adaptive lattice filters in real-time. To achieve real-time processing speeds, algorithm complexity was accommodated by custom designing the computation engines with respect to the lattice data flow. Execution speeds of the computation engines were dramatically increased by a memory architecture that supports efficient addressing and by providing a floating-point ALU with numerous data paths and efficient implementation of division. Performance is further enhanced by pipelining multiple computation engines. In addition, the architecture is flexible enough to support other filter structures and to allow observation of filter variables as they adapt. With this system, various Adaptive Filter algorithms are being tested in real-time implementations of Adaptive Line Enhancers (ALEs) and Adaptive Noise Cancelers (ANCs) in order to characterize their performance and behavior, especially long term stability and the ability to track non-stationary signals.

I. Introduction

Adaptive lattice algorithms have been expected to offer a number of advantages over conventional LMS transversal algorithms including: faster rate of convergence, modular structure, insensitivity to variations in the eigenvalue spread of the input correlation matrix, and automatic system order detection [1,2]. However, the use of adaptive lattice filters for real-time signal processing has been limited. In part this is due to their computational cost and complexity. This paper describes the design of a hardware test platform, called the Lattice Development System (LDS), designed and built by the U.S. Naval Ocean Systems Center (NOSC) to implement adaptive lattice filters for real-time applications. The design, while tailored to implement adaptive lattice filters efficiently, is flexible enough to support other structures such as adaptive transversal filters for comparative performance evaluations.

The recursive nature of adaptive filters makes their implementation in real-time hardware an extremely interesting and challenging research field. Time-domain adaptive algorithms generally require that their filtered output be used in updating the filter's coefficients *before* the next input sample is processed. Thus, the total processing latency must be less than one sample period. Unfortunately, the coefficient updating can often dominate the total processing time thereby limiting the adaptive filter's potential applications. Multiprocessing techniques used for performance enhancement of non-adaptive filters can not be directly applied to adaptive filters due to adaptive filter's lack of a single computational form. For instance, non-adaptive convolution is composed of only a sum-of-products. Adaptive filters usually posses a number of computational forms as well as requiring several different modes of operation such as initialization, adaptation, and order expansion & contraction.

Figure 1 shows the tradeoffs incurred by different methods of implementing adaptive filters. Usually, performance and design complexity are traded against flexibility. The measure of performance in real-time systems is the maximum continuous sampling rate supported by the hardware; the measure of flexibility is how easily modifications can be accomplished. Examples of features one may wish to modify are: update algorithm, filter structure, filter configuration, and filter parameters such as order and time constants.

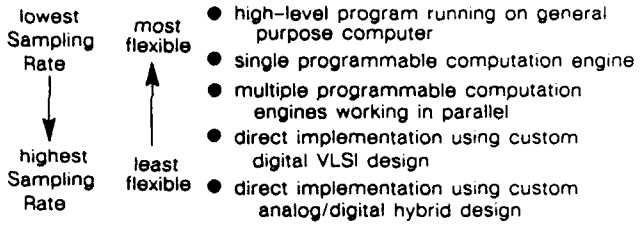

Figure 1: Adaptive Filter Implementation Methods and their Tradeoffs.

The LDS was designed as an adaptive filter test platform to characterize the performance and behavior of both adaptive lattice algorithms and adaptive transversal algorithms in real-time applications. Of particular interest are the long term stability and the ability to track non-stationary signals as a function of filter parameters. To meet the combined needs of high throughput and flexibility, the LDS was built as a linear pipelined array of custom designed, programmable computation engines. Each engine is optimized for implementing adaptive lattice filters using a single 32-bit floating-point ALU with provisions for floating-point division. The LDS system can be configured with up to 10 engines, each having sufficient memory to store the variables for a 1024 stage lattice filter. A system with a full complement of ten computation engines is capable of sustaining the computation of a 1024 stage recursive least square lattice (RLSL) filter [10,11] at a 1.2KHz real-time sample rate.

Other adaptive lattice filter implementations, particularly those in custom VLSI, have been based on linear pipelined processing arrays consistent with the lattice structure [3-5]. One implementation made use of a switched capacitor filter in an analog/digital hybrid approach [6]. Recently, a vectorized adaptive lattice was proposed which allows for even higher degrees of parallelism [7-9]. All of these take advantage of the modularity provided by the local (single lattice stage) error feedback of the lattice.

Section II of this paper is a high level description of the LDS. It includes a brief description of the system's three main hardware functional units, plus the Man-Machine Interface software and other software support tools. In addition, Section II describes the implementation of adaptive filters in a multiple engine LDS configuration. Section III provides more detailed information and insight into the design of the functional unit that is the processing heart of the LDS: the Computation Engine. A sample of the capabilities of the LDS is presented in Section IV.

II. Hardware System Overview

The LDS has three separate functional units: the System Control, the Analog-Digital Interface, and an array of Computation Engines. System performance is increased by using multiple computation engines in a linear pipelined array. The functional units are accessed and configured by a single board mini-computer running a menu driven man-machine interface (MMI). Figure 2 shows the interconnection and the user interface via the mini-computer and a terminal. Once configured by the user, the LDS will run in real-time until stopped by the user. It can easily be reconfigured to compute different adaptive filter algorithms, modify filter parameters, change sample frequency, or change between ALE and ANC. The diagram

Accepted for Publication: PROCEEDINGS OF THE ASILOMAR CONFERENCE ON CIRCIUTS, SYSTEMS, AND COMPUTERS, Pacific Grove, CA, Nov 1990

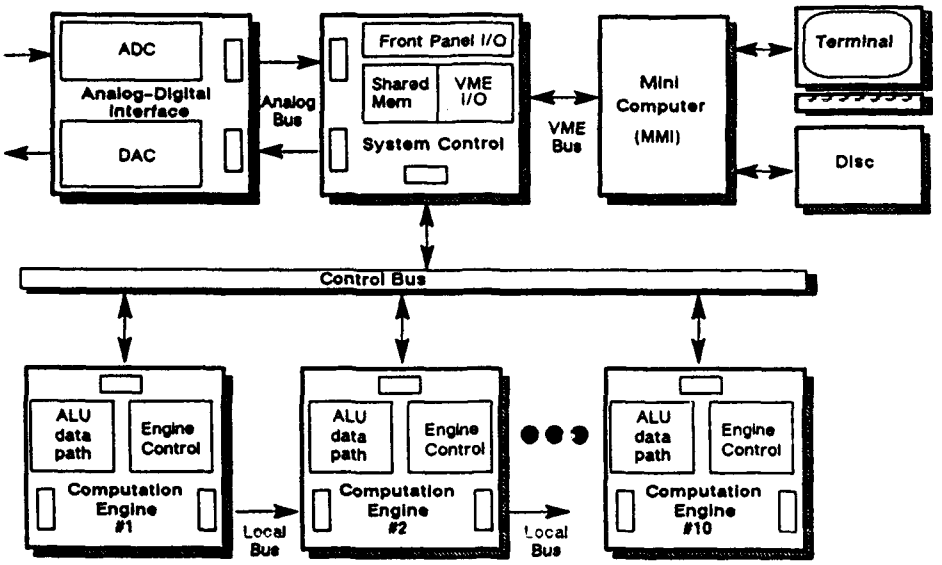

Figure 2: Block Diagram of the NOSC Adaptive Lattice Development System (LDS).

in Figure 3 shows the ALE and ANC adaptive filter configurations and the associated nomenclature for the reference, x(n), primary, d(n), filter output, y(n), and error output, e(n), signals.

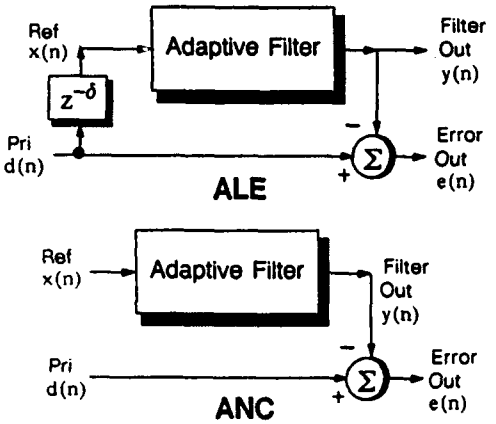

Figure 3: Adaptive Line Enhancer (ALE) and Adaptive Noise Canceler (ANC) Filters.

Real-Time Operation

The System Control unit runs a concurrent operating system which initializes all actions in the LDS. Control data is passed by the MMI to the System Control unit to configure or reconfigure the system. The control unit first down loads the microcode for each engine and then initializes each engine's state. Once the system is completely configured, it will perform the following tasks repeatedly when commanded to run;

1. prompts the A-D Interface unit to acquire 16-bit quantized data,
2. accepts data from the A-D Interface unit and places it in local RAM,
3. passes input data to the engines for filtering,
4. initiates data transfers between Computation Engines,
5. accepts filter output data and other outputs from the engines,
6. sends engine outputs to the A-D Interface unit for conversion to analog output,
7. loops to 1.

All functional units are synchronous and run from a single 8MHz clock distributed throughout the LDS. Engine operations are started synchronously but proceed independently, so they can run different microcode and may terminate operations independently. Once all engines terminate operations in a given filter update cycle, the System Control unit assumes control. The computation cycle repeats with synchronous starts on command from the System Control unit and asynchronous terminations dependent upon computation requirements.

Local RAM on the System Control unit is designed to handle multiple circular buffers so that input data (reference and/or primary) can be delayed up to a combined maximum of 16K data samples before being sent to the engines. This supports many filter configurations including the ALE, where the reference data is a delayed version of the primary data. Another feature of the LDS is a user definable sampling frequency.

Filter Implementations with a Multiple Engine LDS

Both adaptive lattice and adaptive transversal filters can benefit from a multiple engine LDS configuration. The lattice filter's order recursive variables are passed from stage to stage in sequential order. This can result in extremely long processing times when long filter lengths are used. The total processing time can be reduced by a factor of O(P) by using P pipelined computation engines. Since each engine must be pipelined, successive engines will be processing data that is one sample period earlier in time than the engine which supplies its passed variables. An application's required sample rate determines both the number of engines used and the maximum number of stages computed on each engine. Figure 4 shows an example of a three engine LDS computing a 6-stage adaptive lattice filter. At the completion of a lattice update cycle, all order recursive variables are passed between engines via an uni-directional local bus in a synchronous fashion.

Adaptive transversal filters can be implemented on a multiple engine LDS in several ways, even though multiprocessing with them is less straight forward than for adaptive lattice filters due to the global error feedback for coefficient updating [12]. One of the more efficient approaches makes use of the multiple engines configured in a uni-directional data flow ring, as was proposed by Miller et. al. [13]. As an example of this approach, Figure 5 shows the computation of an 8-weight adaptive transversal filter on a four engine LDS. Each engine is responsible for only those calculations involving a section of the complete filter's weights. The process proceeds by each engine computing the convolution sum of its section of the filter. A series of synchronous data passing and summation steps are then performed until each engine has its own identical copy of the filtered output. Next each engine generates its own identical error term from a broadcasted desired signal and updates its own set of weights via an adaptive algorithm such as least mean square (LMS) [14]. The method effectively solves the

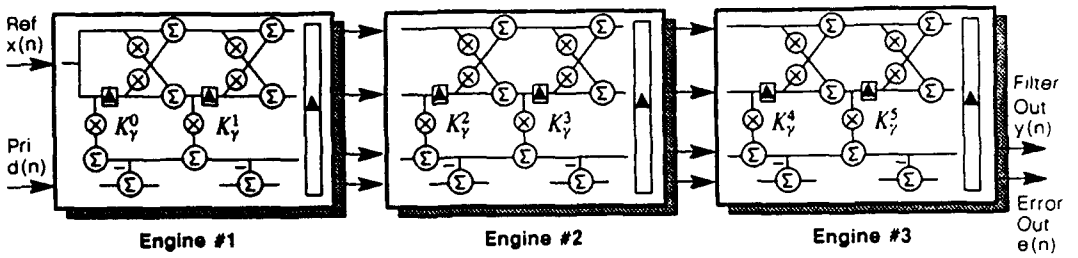

Figure 4: Calculation of a 6-stage Adaptive Lattice Filter in a Three Engine LDS.

global feedback problem by generating the same error in every processor. Each engine has sufficient memory to store the variables for a 4096-weight LMS filter.

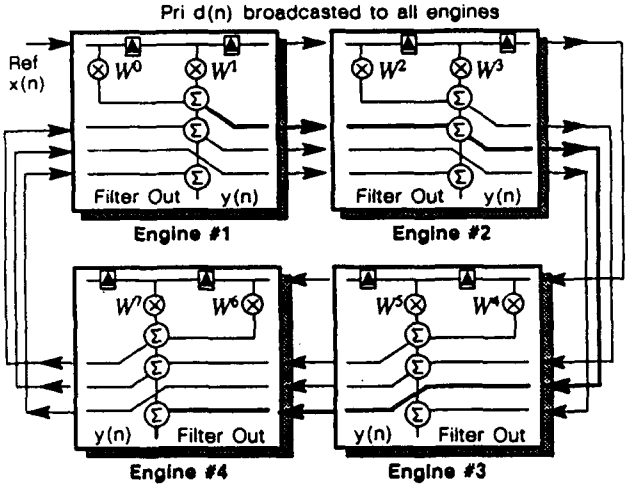

Figure 5: Calculation of an 8-stage Adaptive Transversal Filter in a Four Engine LDS.

Analysis Tools

The LDS software provides the user with many useful features for analyzing adaptive filters. Specific filter coefficients can be selected for observation as they adapt in time, or a *snapshot* of all coefficients can be taken at a specific time period. The coefficients are transferred to data files on the mini-computer's hard disc from which further analysis can be done off-line. The system also has the capability of inputting data directly from a data file or outputting filtered data directly to a file.

III. Computation Engine Design

The Computation Engine is the processing heart of the LDS. It was custom designed to efficiently implement adaptive lattice filters and is composed of two main parts: the microsequencer and the microengine. The engine's operation is controlled by up to 2K microwords down loaded from the System Control unit during the LDS's initialization. The equations that describe adaptive lattice algorithms are structured into groups called stages (see Figure 6). Each stage can have variables which are updated entirely by time-recursions, entirely by order-recursions or by a combination of both time and order-recursions. Adaptive lattice algorithms such as the RLSL [10,11] and the stochastic gradient lattice [15] use both time and order recursions to save computations. In addition, these adaptive lattice algorithms require multiple divisions per stage. The following features were incorporated in order to achieve real-time computation speeds for adaptive lattice filters:

- use of a 32-bit floating-point processing chip with a latency of only one clock cycle,
- efficient implementation of floating-point division,
- separate memories for time-recursion and order-recursion variables,
- implementation of a floating-point & integer comparator,
- provision of many data paths to support all of the above,
- extension of data paths across multiple computational engines,
- increased data bandwidth by using uni-directional local busses to link neighboring engines,
- use of a simple microsequencer and a horizontal microword.

The utility of these features is described in the next two sections

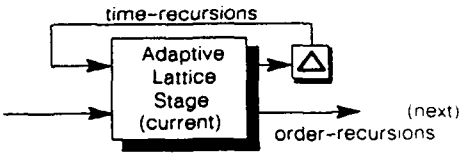

Figure 6: Data Flow for Single Adaptive Lattice Stage Computations.

Microengine Arithmetic Structure and Data Paths

Figure 7 shows the data paths in the microengine. The Advanced Micro Device's AM29325 [16] was the floating-point processor chosen for two main reasons: it can compute arithmetic operations in a single clock cycle and it supports fast inversion instructions. A short pipeline depth is desirable when implementing tightly recursive algorithms, such as RLSL, which often require the result of one computation for the very next computation. Inversions are computed with a Newton-Raphson iteration technique using a first approximation seed sufficiently large to require only one iteration (a total of three floating-point operations) to achieve full 32-bit floating-point precision. This technique has quadratic convergence properties and the only additional hardware needed is an inverse seed PROM. In parallel with the AM29325 is a floating-point & integer comparator which supports such things as variable bounding and lattice filter order control.

Numerous data paths allow for the efficient moving of data between the AM29325, registers, and memory. All data paths and operations are controlled by individual terms in the microcode. A 96-bit horizontal microword is used which allows complete flexibility and eliminates decoding time. One additional data path not shown in Figure 7, but frequently used, is an internal wrap around data path inside the AM29325. The microengine is programmed with a user defined assembler, aided by using a reservation chart to keep track of the multiple data paths and concurrent operations.

Automatic system order detection is a unique capability of adaptive lattice filters. Control of the length of the lattice filter is necessary to minimize filter generated noise and insure stability of the final error, e(n), and filter output, y(n). Rapid variation of filter order is possible with non-stationary signals so order control must function in real-time. The forward and backward prediction residuals of each successive stage must be compared to a threshold as they are computed to determine if another stage is needed (or allowed) for the current update. The value of the threshold is defined by the user, based on the engine's arithmetic precision and the exponential windowing of the data. Control must span across engine boundaries and accommodate the time displacement at those boundaries. This is done with a special flag set by each engine which is checked and cleared on the next iteration by its downstream target engine

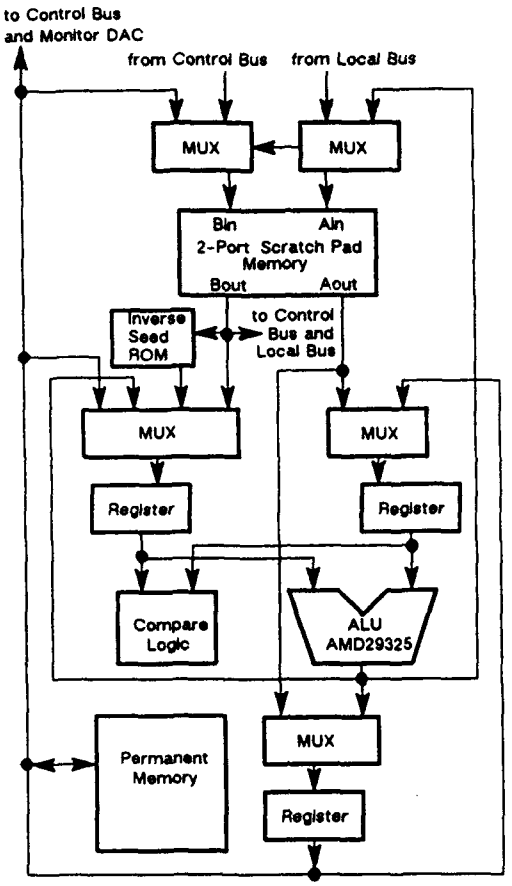

Figure 7: The Computation Engine's Microengine Data Path.

Microengine Memory Structure

Two separate data memories are available to the AM29325, one is tailored to support time-recursions, the other order-recursions. The Permanent Memory (PM) is addressable in blocks corresponding to lattice stages for ease of storage and retrieval during time-recursions. Eight variables can be stored in each PM block as seen in Figure 8. The structure is 1K-stages x 8-words x 32-bits allowing a maximum of a 1024 stage lattice filter per engine. The PM uses a block addressing scheme where each block is selected by a stage number counter. The block addressing greatly simplifies the engine's microcode during the computation of a lattice stage without sacrificing the use of simple addressing schemes for transversal filters.

The Scratch Pad Memory (SPM) in addition to being a temporary storage area for data transfers and intermediate results, can be structured to operate in a "ping-pong" fashion between the upper and lower halves to allow the overwriting of order-recursion variables that are no longer needed. During the computation of a lattice filter, variables are passed between the two SPM halves with the *current* stage overwriting variables passed from a previous stage. Each SPM half contains 32-words x 32-bits of dual-ported memory. With *current/next* addressing, there is no need to keep track of the absolute physical addresses being used which saves machine cycles and engine instructions. In addition, it is easier to conceptualize order recursion variables as *current* and *next* (Figure 6). Special address control logic switches the physical location of the order recursion variables in a manner that is transparent to the software (and the programmer). The result is a very efficient use of memory space and a simple, efficient addressing scheme. For the computation of transversal filters, the SPM is configured by software as a single 64-word x 32-bit dual-port memory (see Figure 9). Memory locations in both the SPM and the PM can be accessed in a single clock cycle at the same time.

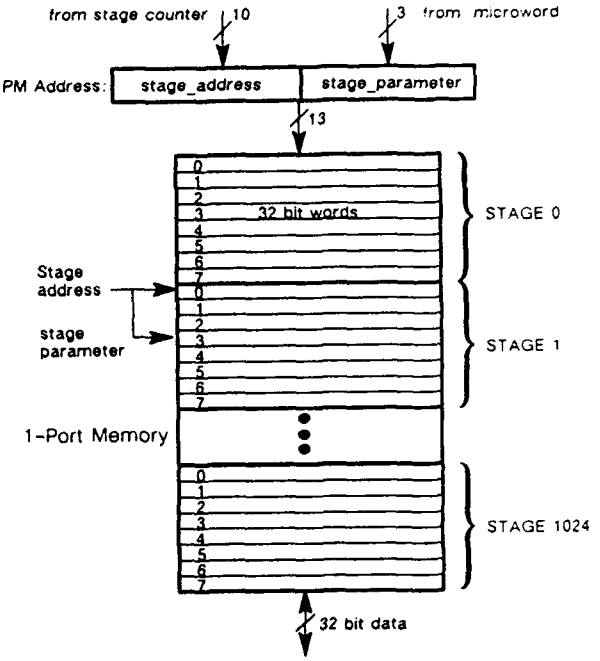

Figure 8: Engine Board Permanent Memory Configuration (1K x 32-bits x 8).

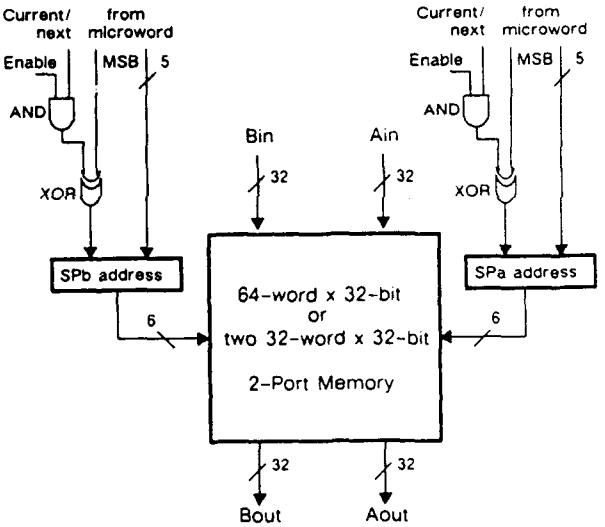

Figure 9: Engine Board Scrach Pad Memory Configuration (64 x 32-bits or two 32 x 32-bit halves)

IV. LDS Test Data

As of this writing the following adaptive filter algorithms have been programmed: LMS transversal [14], Block LMS transversal [17], Stochastic Gradient Lattice [15], Recursive Least Squares Lattice [10,11] and Direct Coefficient Updating RLSL [18]. Testing with the LMS algorithm began in March of 1989 while testing with the RLSL began in June of 1989. The other algorithms have been included in 1990. A photograph of the LDS is provided in Figure 10. Figure 11 is a photograph of the two boards comprising a Computation Engine.

As an example of LDS operation, the time series outputs of LMS and RLSL ANCs are shown for comparison in Figures 12 and 13 respectively. In both cases a single sinusoid is being canceled from the primary input. Note the characteristic exponential adaptation of LMS, and the nearly instantaneous adaptation of the RLSL

V. Conclusion

This paper has presented the design of a real-time adaptive filter development system. The system was custom built for the purpose of studying the performance of lattice algorithms in real world environments and comparing them to other adaptive algorithms. To this end, special hardware and software features were incorporated into the design to maximize both performance and flexibility. In addition, provisions to allow the observation of filter variables during adaptation were incorporated to aid analysis. The system is typically configured as a linear array of programmable engines to enhance execution speeds. A description of the architecture and design details were presented along with sample test data.

Acknowledgments

Special thanks are offered to Dr. Mark Shensa for his patience and expertise in helping us fully understand and appreciate the elegance of the RLS Lattice algorithm, and, to our sponsor for continued support through the years.

References

[1] S. Haykin, *Adaptive Filter Theory*, Prentic-Hall, New Jersey, 1986.

[2] M.L. Honig and D.G. Messerschmitt, *Adaptive Filters: Structures, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1984.

[3] H.M. Ahmed, M. Morf, D.T. Lee and P.H. Ang, "A VLSI Speech Analysis Chip Set Based on Square-Root Normalized Ladder Forms," *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '81)*, pp. 648-653, 1981.

[4] M.J. Rutter, P.M. Grant, D. Renshaw and P.B. Denyer, "Design and Realization of Adaptive Lattice Filters," *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '83)*, paper 1.6, pp. 21-24, 1983.

[5] K. Konstantinides and N. Kanopoulos, "Design Alternatives for Adaptive Digital Lattice Filters and a New Bit-Serial Architecture," *IEEE Trans. on Circuits and Systems*, vol. CAS-34, no. 7, pp. 737-742, July 1987.

[6] R.D. Fellman and R.W. Brodersen, "A Switched-Capacitor Adaptive Lattice Filter," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-31, no. 1, pp. 294-304, Feb. 1983.

[7] T.H.-Y. Meng and D.G. Messerschmitt, "Arbitrararily High Sampling Rate Adaptive Filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, no. 4, pp. 455-470, April 1987.

[8] T.H.-Y. Meng, R.W. Brodersen and D.G. Messerschmitt, "A Clock-Freee Chip Set for High-Sampling Rate Adaptive Filters," *Journal of VLSI Signal Processing*, vol. 1, pp. 345-365, 1990.

[9] M.D. Meyer and D.P. Agrawal, "Adaptive Lattice Filter Implementations on Pipelined Multiprocessor Architectures," *IEEE Trans. on Communications*, vol. COM-38, no. 1, pp. 122-124, Feb. 1990.

[10] E.H. Satorious and J.D. Pack, "Least Squares, Adaptive Lattice Algorithms," *NOSC Technical Report #423*, April 1979.

[11] M.J. Shensa, "Recursive Least-Squares Lattice Algorithms: A Geometrical Approach," *IEEE Trans. Automatic Control*, vol. AC-26, pp. 695-702, June 1981.

[12] V.B. Lawrence and S.K. Tewksbury, "Multiprocessor Implementation of Adaptive Digital Filters," *IEEE Trans. on Communications*, vol. COM-31, no. 6, pp. 826-835, June 1983.

[13] T.K. Miller, S.T. Alexander and L.J. Faber, "An SIMD Multiprocessor Ring Architecture for the LMS Adaptive Algortihm," *IEEE Trans. on Communications*, vol. COM-34, no. 1, pp. 89-92, Jan. 1986.

[14] B. Widrow, J. Glover, J. McCool, J. Kaunitz, C. Williams, R. Hearn, J. Zeidler, E. Dong, Jr. and R. Goodin, "Adaptive Noise Cancelling: Principles and Applications," *Proc. IEEE*, vol. 63, pp. 1692-1716, Dec. 1975.

[15] L.J. Griffiths, "An Adaptive Lattice Structure for Noise-Cancelling Applications," *Proc. IEEE Int. Conf. on ASSP*, Tulsa, Ok., pp. 87-90, Apr. 1978.

[16] Advanced Micro Devices, "AM29325 32-bit Floating-Point Processor," *Publication #05621*, Feb. 1989.

[17] G.A. Clark, S.K. Mitra, and S.R. Parker, "Block Implementation of Adaptive Digital Filters," *IEEE Trans. on Circuits and Systems*, vol. CAS-28, No. 6, pp. 584-592, June 1981.

[18] F. Ling, D. Manolakis, and J.G. Proakis, "Numerically Robust Least-Squares Lattice-Latter Algorithms with Direct Updating of the Reflection Coefficients," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 4, pp. 837-845, Aug. 1986.

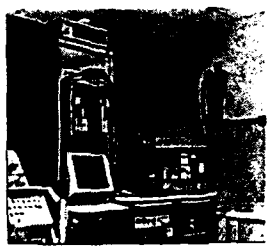

Figure 10: Photograph of the NOSC Adaptive Lattice Development System (LDS).

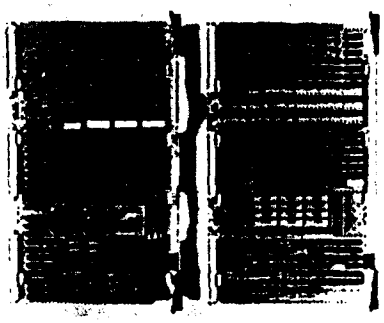

Figure 11: Photograph of the Computation Engine Boards

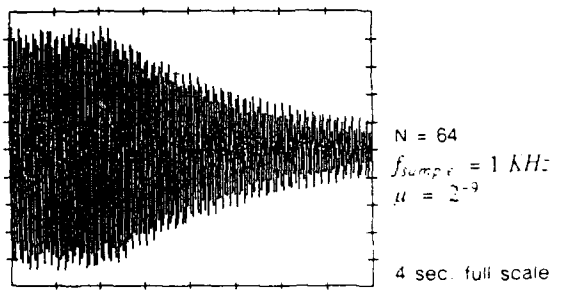

Figure 12: LMS Adaptive Noise Cancellation of a Single Sinusoid.

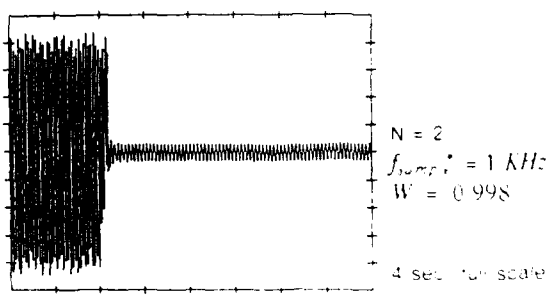

Figure 13: RLSL Adaptive Noise Cancellation of a Single Sinusoid